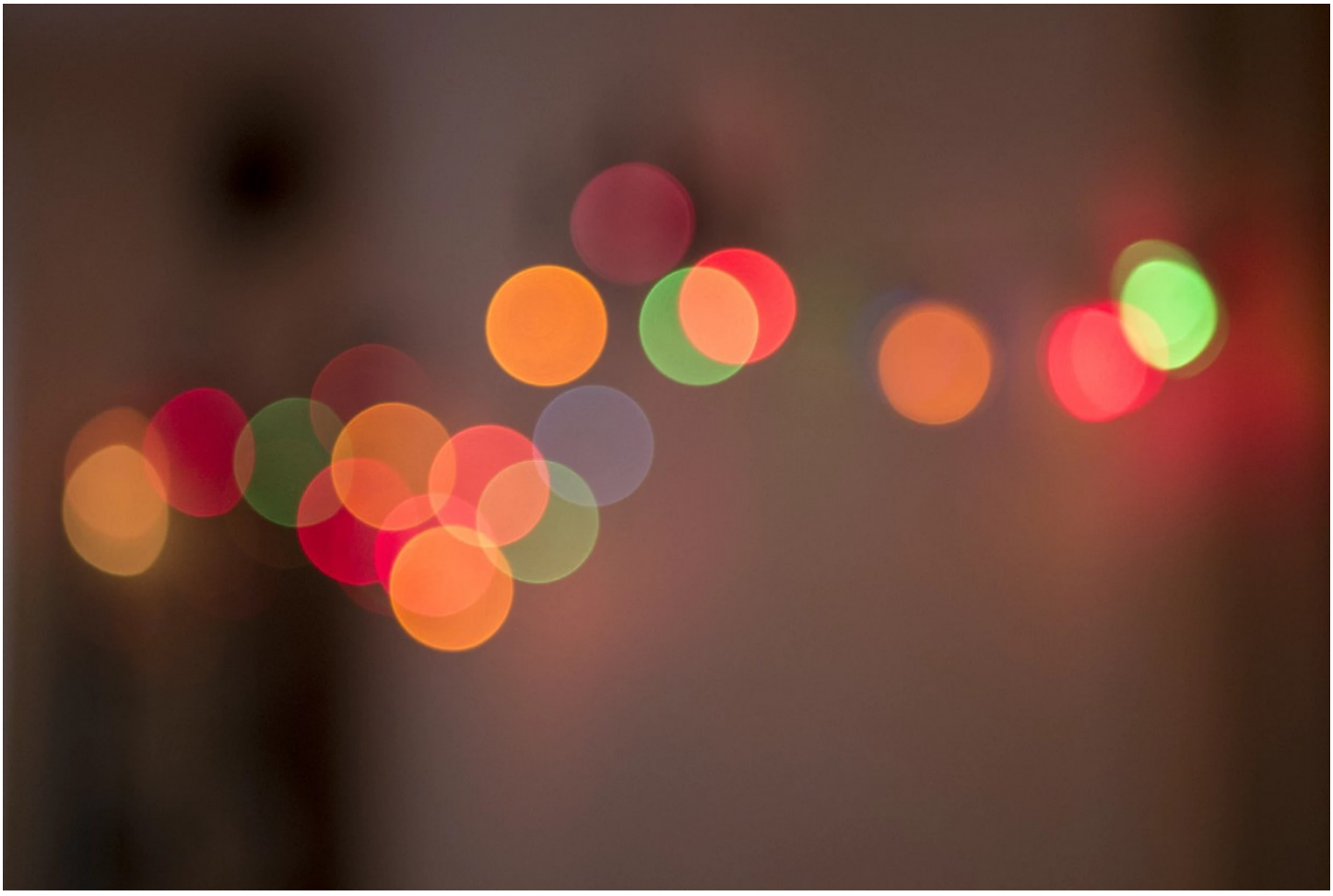
JavaScript ES6— The Spread Syntax (...)

"Expand" your JavaScript knowledge with the Spread Syntax

Brandon Morelli Follow
Feb 6, 2018 · 4 min read



The Spread Syntax

- The spread syntax is simply three dots: ...
- **It allows an iterable to expand in places where 0+ arguments are expected.**

Definitions are tough without context. Lets explore some different use cases to help understand what this means.

Example #1 — Inserting Arrays

Take a look at the code below. In this code, we **don't** use the spread syntax:

Above, we've created an array named `mid`. We then create a second array which contains our `mid` array. Finally, we log out the result. What do you expect `arr` to pring? Click **run** above to see what happens. Here is the output:

```
[1, 2, [3, 4], 5, 6]
```

Is that the result you expected?

By inserting the `mid` array into the `arr` array, we've ended up with an array within an array. That's fine if that was the goal, but what if want only a single array with the values of 1 through 6? To accomplish this, we can use the spread syntax! Remember, the spread syntax allows the elements of our array to **expand**.

Lets look at the code below. Everything is the same — except we're now using the spread syntax to insert the `mid` array into the `arr` array:

And when you hit the run button, here's the result:

```
[1, 2, 3, 4, 5, 6]
```

Awesome!

Remember the spread syntax definition you just read above? Here's where it comes into play. As you can see, when we create the `arr` array and use the spread operator on the `mid` array, instead of just being inserted, the `mid` array *expands*. This expansion means that each and every element in the `mid` array is inserted into the `arr` array. Instead of nested arrays, the result is a single array of numbers ranging from 1 to 6.

Example #2 — Math

JavaScript has a built in math object that allows us to do some fun math calculations. In this example we'll be looking at `Math.max()` . If you're unfamiliar, `Math.max()` returns the largest of zero or more numbers. Here are a few examples:

```
Math.max();
// -Infinity

Math.max(1, 2, 3);
// 3

Math.max(100, 3, 4);
// 100
```

As you can see, if you want to find the maximum value of multiple numbers, `Math.max()` requires multiple parameters. You unfortunately can't simply use a single array as input. Before the spread syntax, the easiest way to use `Math.max()` on an array is to use `.apply()`

```
var arr = [2, 4, 8, 6, 0];

function max(arr) {
  return Math.max.apply(null, arr);
}

console.log(max(arr));
```

Save on RunKit Node 10

```
8
undefined
```

It works, it's just really annoying.

Now take a look at how we do the same exact thing with the spread syntax:

```
var arr = [2, 4, 8, 6, 0];
var max = Math.max(...arr);

console.log(max);
```

Save on RunKit Node 10

```
8
undefined
```

Instead of having to create a function and utilize the `apply` method to return the result of `Math.max()` , we only need two lines of code! The spread syntax expands our array elements and inputs each element in our array individually into the `Math.max()` method!

Example #3 — Copy an Array

In JavaScript, you can't just copy an array by setting a new variable equal to already existing array. Consider the following code example:

```
var arr = ['a', 'b', 'c'];
var arr2 = arr;

console.log(arr2);
```

Powered by RunKit Node 10

When you press **run**, you'll get the following output:

```
['a', 'b', 'c']
```

Now, at first glance, it looks like it worked — it looks like we've copied the values of `arr` into `arr2`. But that's not what has happened. You see, when working with objects in javascript (arrays are a type of object) we assign by reference, not by value. This means that `arr2` has been assigned to the same reference as `arr`. In other words, anything we do to `arr2` will also affect the original `arr` array (and vice versa). Take a look below:

```
var arr = ['a', 'b', 'c'];
var arr2 = arr;

arr2.push('d');

console.log(arr);
```

Powered by RunKit Node 10

Above, we've pushed a new element `d` into `arr2`. Yet, when we log out the value of `arr`, you'll see that the `d` value was also added to that array:

```
['a', 'b', 'c', 'd']
```

No need to fear though! We can use the spread operator!

Consider the code below. It's almost the same as above. Instead though, we've used the

spread operator within a pair of square brackets:

```
var arr = ['a', 'b', 'c'];
var arr2 = [...arr];

```

Powered by RunKit Node 10

Hit run, and you'll see the expected output:

```
['a', 'b', 'c']
```

Above, the array values in `arr` expanded to become individual elements which were then assigned to `arr2`. We can now change the `arr2` array as much as we'd like with no consequences on the original `arr` array:

Again, the reason this works is because the value of `arr` is expanded to fill the brackets of our `arr2` array definition. Thus, we are setting `arr2` to equal the individual values of `arr` instead of the reference to `arr` like we did in the first example.

Bonus Example — String to Array

As a fun final example, you can use the spread syntax to convert a string into an array. Simply use the spread syntax within a pair of square brackets:

Pretty cool, right?

Closing Notes:

Thanks for reading! If you're ready to finally learn Web Development, check out: **The**

If you're working towards becoming a better JavaScript Developer, check out: **Ace Your Javascript Interview — Learn Algorithms + Data Structures**.

I publish 4 articles on web development each week. Please consider **entering your** if you'd like to be added to my once-weekly email list, or follow me on

If this post was helpful, please click the clap 👏 button below a few times to show your support! ⬇⬇